VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

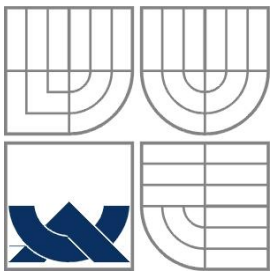
ROZPOZNÁNÍ FIGUR NA ŠACHOVNICI Z FOTOGRAFIE NA MOBILNÍM ZAŘÍZENÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

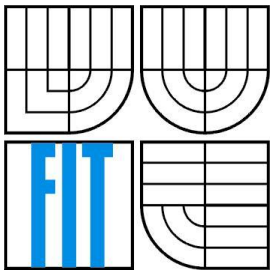
AUTOR PRÁCE
AUTHOR

ROMAN TÁBI

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁNÍ FIGUR NA ŠACHOVNICI Z FOTOGRAFIE NA MOBILNÍM ZAŘÍZENÍ

RECOGNITION OF PIECES ON CHESSBOARD FROM PHOTOGRAPHY ON MOBILE DEVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN TÁBI

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAKUB SOCHOR

BRNO 2015

Abstrakt

Tato práce řeši mobilní aplikaci vytvořenou pro platformu Android, která rozpoznává figury z fotografie šachovnice. Aplikace vybere z galerie zařízení fotografii, kterou pak spracuje a určí obsah jednotlivých políček, tedy zda obsahuje černou figuru, bílou figuru, a nebo je políčko prázdné. Problém je řešen ve dvěch fázách. První je detekce šachovnicových přímek pro určení polohy jednotlivých políček, druhá pak detekce figury.
Výsledkem této práce je funkční Android aplikace.

Abstract

The aim of this bachelor thesis is a mobile application for Android platform, which recognises chessboard pieces from photography of the chessboard. Application uses photography from gallery and determine the content of every single chessboard cell, thus if it contains black or white piece, or if it is empty.
The task is solved in two stages. First is the detection of chessboard lines to determine the position of chessboard cells, second is the detection of chessboard pieces.
The result of this thesis is working Android application.

Klíčová slova

šachy, android, detekce, Canny, Sobel, šachovnice

Keywords

chess, android, detection, Canny, Sobel, chessboard

Citace

Tábi Roman: Rozpoznání figur na šachovnici z fotografie na mobilním zařízení, bakalářská práce, Brno, FIT VUT v Brně, 2015

Rozpoznání figur na šachovnici z fotografie na mobilním zařízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením p. Ing. Jakuba Sochora. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Roman Tábi
20. května 2015

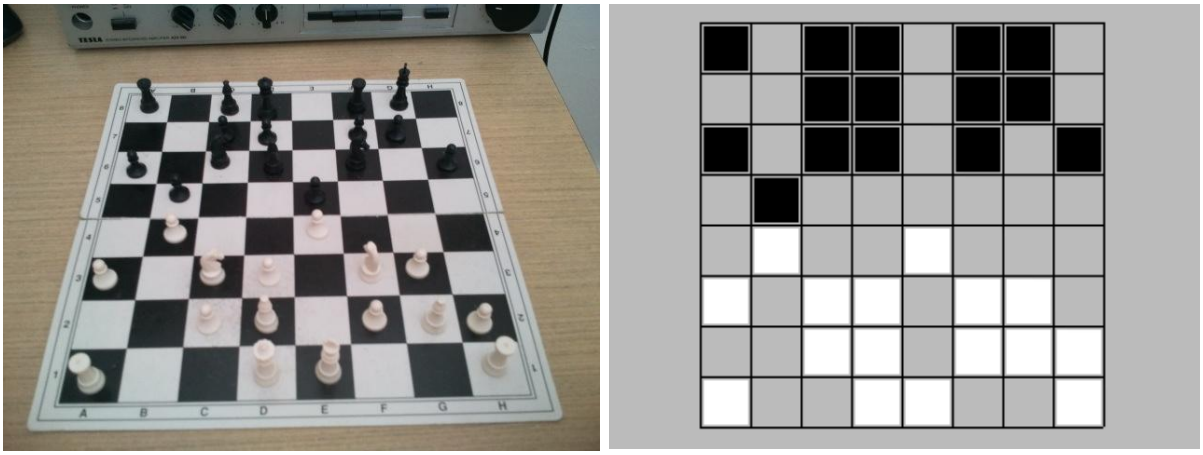
Obsah

1 Úvod

Šach je hra kráľov, ktorá so sebou nesie kus histórie. Samotný strategický princíp a počet možných situácií, kombinácií a postupov z neho robí výnimočnú a v týchto smeroch tažko prekonateľnú spoločenskú hru.

Spájať historickú klasiku s modernými technológiami je vždy zaujimavá záležitosť a táto práca pojednáva o jednej z nich. Cieľom tejto bakalárskej práce je vytvoriť mobilnú aplikáciu, ktorá bude schopná plne automaticky detekovať figúrky na fotografii šachovnice. Aplikácia má určiť pozíciu a farbu jednotlivých figúrok. Postup detekcie spočíva v niekoľkých krokoch, ako detekcia rohov, priamok, výpočet súradníc políčok a rozhodovanie o obsahu konkrétneho políčka šachovnice.

Práca je štruktúrovaná do kapitol, v ktorých sú popísané existujúce riešenia niektorých častí, použité postupy, navrhnutý a implementovaný systém detekcie ako aj vyhodnotenie úspešnosti jednotlivých detektorov.

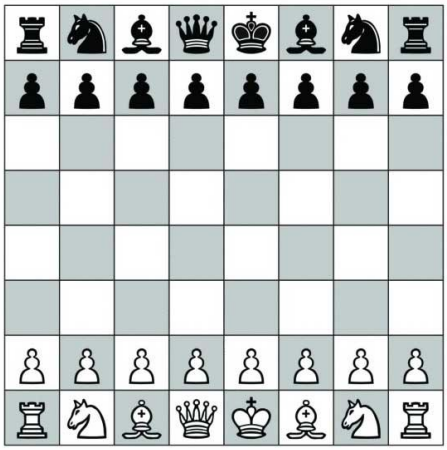


Obrázok 1.1 – Šachovnica a detekované rozloženie

2 Šach

Šach je strategická hra pre dvoch hráčov, spočívajúca v taktickom ťahaní figúrkami po šachovnici. Cieľom hry je dať súperovi mat, čo je situácia, kedy je napadnutý kráľ a z jeho strany nie je možné tejto situácii zabrániť. Hrací komplet pozostáva z hracej dosky – šachovnice, a 32 figúrok, 16 figúrok jednej farby a ďalších 16 druhej farby.

Šachovnicu tvorí 8 stĺpcov a 8 riadkov striedajúcich sa čiernych a bielych štvorcov tak, že dva štvorce rovnakej farby sa dotýkajú vrcholmi, nikdy však hranami. Ľavý dolný roh šachovnice je čierny.



Obrázok 2.1 – Šachovnica a základné rozloženie

(Zdroj: http://www.readersdigest.com.au/userfiles/chess-board.jpg)

Typy figúrok:

Ľahké figúry:

Pešiak

Ťažšie figúry:

Jazdec, Strelec	(hodnota troch pešiakov)
Veža	(hodnota piatich pešiakov)
Kráľovná	(hodnota devietich pešiakov)

Každý hráč má na začiatku hry k dispozícii 16 figúrok usporiadaných nasledovne:

Z pohľadu bieleho

Druhá línia: 8 pešiakov

Prvá línia: Veža, Jazdec, Strelec, Kráľovná, Kráľ, Strelec, Jazdec, Veža

Z pohľadu čierneho

Druhá línia: 8 pešiakov

Prvá línia: Veža, Jazdec, Strelec, Kráľ, Kráľovná, Strelec, Jazdec, Veža

- *Platí pravidlo - kráľovná stojí na vlastnej farbe (farba kráľovny je rovnaká ako políčko pod ňou)*

Pravidlá šachu definujú prípad, keď sa pešiak nemôže nachádzať na prvej/poslednej línii, ale v prípade prechodu pešiaka až na poslednú líniu, má hráč možnosť vymeniť tohto pešiaka za niektorú z ťažších figúr (nie však kráľa). Týmto spôsobom môže nastať situácia, kedy má hráč v poli viac rovnakých figúrok toho istého druhu (väčšinou viac kráľovien z dôvodu výberu najsilnejšej možnej figúrky). Maximálny možný počet figúrok jednej farby je však 16, minimálny len jedna, konkrétne kráľ.

3 Existujúce riešenia detekcie šachovnice

3.1 Harris Corner detector

Podstatou Harrisovho detektoru rohov [1] je vypočítavanie rozdielov hodnoty šedej každého pixlu v zadanom obraze. Je založený na lokálnej autokorelačnej funkcií (matici) signálu; kde lokálna autokorelačná funkcia meria lokálne zmeny signálu s malým posunom do rôznych smerov. Najprv sa vypočíta autokorelačná matica M, potom sa rozhodne, ktorý bod je kandidátny podla jeho vlastnej hodnoty. Ak sú všetky hodnoty väčšie ako hodnota prahu, berieme prah ako kandidátny bod.

Definujeme funkciu vhodnosti rohu v Harrisovom detektore nasledovne:

$$C = DetM - K \times (trM)^2 \tag{1}$$

Kde $DetM = \lambda_1 \times \lambda_2$, $trM = \lambda_1 + \lambda_2$, $K \in \langle 0.04, 0.06 \rangle$, λ_1 a λ_2 sú vlastné čísla matice M

Harrisov detektor zahŕňa tieto špecifické kroky:

1. Výpočet autokorelačnej matice každého pixlu

   $M = \begin{bmatrix} A & D \\ D & B \end{bmatrix}$, kde $A = \left(\frac{\partial I}{\partial x}\right) \otimes w$, $B = \left(\frac{\partial I}{\partial y}\right) \otimes w$, D= $\left(\frac{\partial I}{\partial x}\frac{\partial I}{\partial y}\right) \otimes w$ (2)

   $\otimes$ predstavuje konvolúciu, w je Gaussová funkcia,
2. Výpočet funkcie vhodnosti rohu $C(x, y)$ pre každý pixel (x, y), rov. (1)
3. Výber pixelu s najvyššou hodnotou funkcie vhodnosti rohu ako kandidáta rohu
4. Nastavenie hotnoty prahu, a eliminovanie bodu s $C(x, y)$ hodnotou menšou ako hodnota prahu

3.2 SUSAN detection algorithm

SUSAN algoritmus [2] je založený na kontraste jasu a nepotrebuje vypočítavať rozdiely v obraze. Z toho dôvodu nie je potrebné eliminovať šum a výpočetná rýchlosť je vyššia oproti Harrisovmu algoritmu. Má však obmedzenia pri detekcii rohov v tvare „X“.

Algoritmus:

Ako je ukázané na Obr. 3.1, ak hodnota šedej každého pixlu v rámci šablóny je porovnaná s hodnotou šedej pixlu v strede šablóny, zistíme, že vždy má určitá časť v rámci šablóny približne rovnakú hodnotu šedej ako hodnota prostredného pixlu (nuclear pixel gray) - USAN. SUSAN algoritmus rozlišuje, či aktuálny bod je bod v nútri plochy, na okraji plochy alebo vrchol podla pozície každého bodu v rámci USAN regiónu.

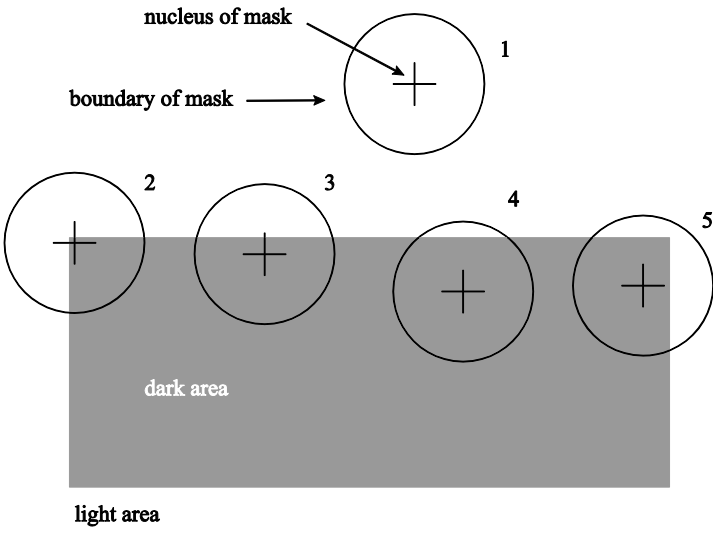
V každej pozícii, hodnota šedej každého pixlu v rámci šablóny je porovnaná s hodnotou prostredného pixlu:

$$C(x_0, y_0; x, y) = e^{-\left[\frac{f(x_0,y_0)-f(x,y)}{T}\right]^6} \tag{3}$$

$$S(x_0, y_0) = \sum_{(x,y)\in N(x,y)} C(x_0, y_0; x, y) \tag{4}$$

Vo výrazoch, (x_0, y_0) je pozícia aktuálneho bodu, (x, y) je pozíciainého bodu v rámci šablóny, $f(x, y)$ vyjadruje hodnotu šedej, T je dopredu zadaný prah šedej, C je výstup a $S(x_0, y_0)$ vyjadruje USAN región. V ďalšom kroku sa porovná USAN región $S(x_0, y_0)$ a dopredu zadaná hodnota prahu G. Pri detekcii rohu by USAN región mal byť menší ako polovica plochy šablóny. Navyše by táto hodnota mala byť lokálnym minimom v oblasti rohu.

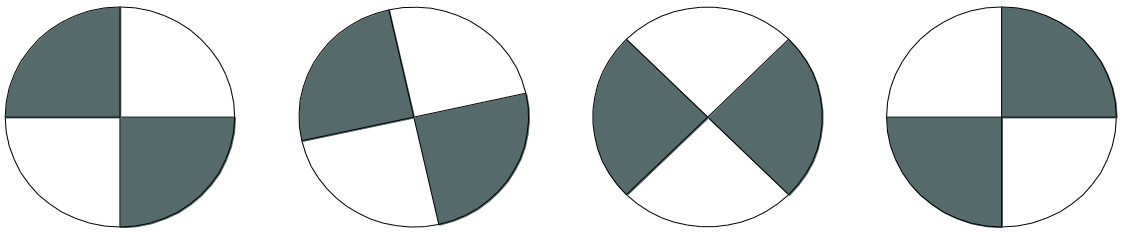
$$R(x_0, y_0) = \begin{cases} G - S(x_0, y_0) & if\ G > (x_0, y_0) \\ 0 & others \end{cases} \tag{5}$$



Obr. 3.1 – SUSAN algoritmus, pozície šablóny

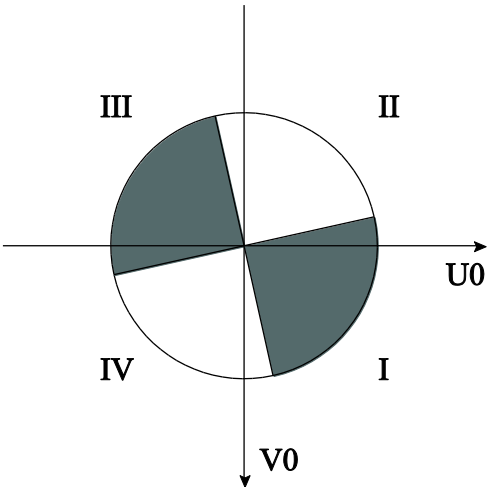
3.3 Improved SUSAN algorithm for chessboard corner detection

SUSAN algoritmus pracuje s USAN oblasťou, ale neberie do úvahy jej tvar. Keď je použitý na detekciu rohu šachovnice, nemôže efektívne rozpoznať roh z okrajového bodu, čo rieši vylepšený SUSAN algoritmus [3].



Obr. 3.2 – roh šachovnice pod kruhovou šablónou USAN oblasti.

Podľa geometrie USAN regiónu sa šachovnicový roh podobá tvaru „X“. Ako je ukázané na Obr. 3.3, nakreslením osí U0 a V0 cez roh ako stred, rozdelíme oblasť na štyri časti. Nezáleží, ako je šachovnica otočená z dôvodu jej geometrickej súmernosti, na Obr. 3.3, I a III oblasť, a II a IV oblasť USAN regiónu sú zhodné alebo blízko podobné, z čoho sa určí poloha rohu šachovnice.



Obr. 3.3 – Analýza geometrickej štruktúry rohu šachovnice

Algoritmus

a) Položíme šablónu na pozíciu každého pixlu, porovnáme hodonty šedej každého pixlu v rámci šablóny s hodnotou prostredného pixlu avypočítame podľa vzťahov (3) a (4).

b) Vypočítame USAN oblasť $S(x_0, y_0)$ s prahovou hodnotou G (polovica najväčšej oblasti USAN regiónu).

Pseudokód:

```
1.  if(G – S(x0,y0) > t1)
2.  {   // potencionálny roh
3.      Houxuan[i][j] = 1;      }
4.  // podmienka rozdielu USAN oblasti a polovice oblasti šablóny
5.  else if(| G – S(x0,y0) | < t2)
6.  {   // podmienka symetrie alebo blízkej podobnosti
7.      if(| SI – SIII | < t3 && | SII – SIV | < t3)
8.          // symetrický roh typu „X“
9.          Houxuan[i][j] = 1;      }
```

t1, t2, t3 sú preddefinované prahové hodnoty, SI, SII, SIII, SIV predstavujú I, II, III, IV oblasti USAN regiónu na Obr. 3.3.

c) Rozdiely hodnôt šedej sú veľké vo všetkých smeroch, preto vyberieme roh ďalej od kandidátneho podľa nasledujúcej funkcie:

$$aver(x_0, y_0) = \frac{1}{\pi r^2} \sum_{(x,y) \in N(x,y)} I(x, y) \qquad (6)$$

$aver(x_0, y_0)$ je priemerná hodnota šedej v rámci šablóny, (x_0, y_0) predstavuje stred a r polomer. $I(x, y)$ je hodnota šedej.

Výsledný roh získame funkciou:

$$R(x_0, y_0) = \sum_{(x,y) \in N(x,y)} (I(x, y) - aver(x_0, y_0))^2 \qquad (7)$$

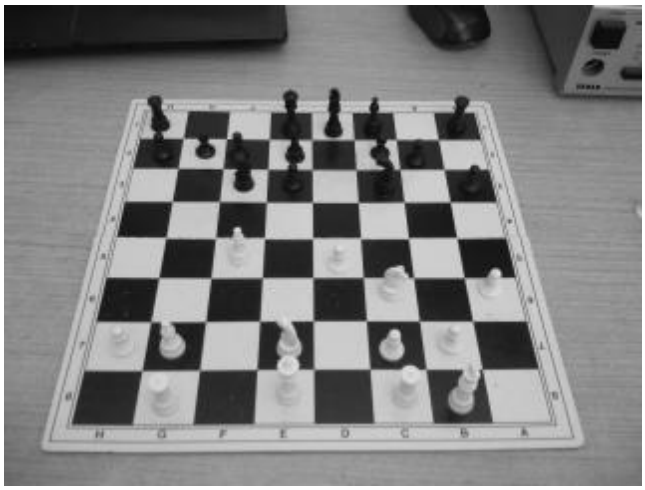
Ak je $R(x_0, y_0)$ väčšie ako hodnota prahu, (x_0, y_0) je roh šachovnice.

4 Použité algoritmy

4.1 Konverzia farebného priestoru

Prevod 24 bit farebného obrazu do šedej je znamená, že každý pixel prevádzáného obrazu bude obsahovať len informáciu o intenzite. Takýto obraz je tvorený iba odtieňmi šedej farby, od čiernej pri najslabšej intenzite po bielu pri najsilnejšej intenzite [3]. Samotný prevod je realizovaný prevodom každého pixlu podľa vzorca

$$I = 0.299R + 0.587G + 0.114B \tag{8}$$

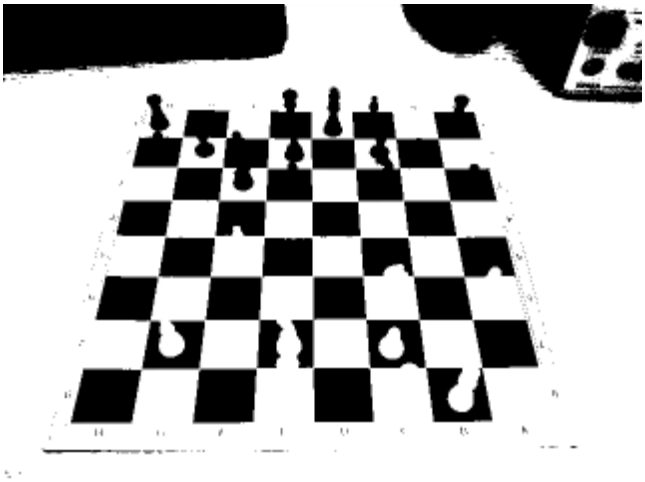


Obr 4.1 – aplikovanie konverzie farebného priestoru

4.2 Prahovanie

Jednoduchá a rýchla metóda prevodu obrazu odtieňu šedej na obraz čiernobiely. Vhodná na použitie na šachovnici z dôvodu vysokého kontrastu jednotlivých políčok. Hodnoty šedej jednotlivých pixlov sa porovnávajú s prahom, ak je hodnota menšia ako prah, výsledný pixel bude čierny, ak je hodnota vyššia ako prah, výsledný pixel bude naopak biely.

$$I(x,y) = \begin{cases} max.hodnota, ak\ hodnota\ šedej(x,y) > prah \\ 0, inak \end{cases} \tag{9}$$



Obr 4.2 – aplikovanie prahovania

4.3 Canny hranový detektor

Canny hranový detektor [4] je derivát Gaussovho hranového detektoru a približuje sa operátoru, ktorý optimalizuje odstup signálu od šumu. Canny hranový detektor sa dá zhrnúť do nasledovného zápisu. Nech $I[i,j]$ značí obraz. Výsledok konvolúcie obrazu a Gaussového vyhladzovacieho filtru pri použití oddeliteľného filtrovania je pole vyhladených dát,

$$S[i,j] = G[i,i;\,\sigma] * I[i,j], \tag{10}$$

kde σ je rozšírenie Gaussovho filtru a udáva stupeň vyhladenia.

Gradient vyhladeného poľa $S[i,j]$ môže byť vypočítaný pomocou 2×2 $first - differnce$ priblížení ([4] Sekcia 5.1) pre vytvorenie dvoch polí $P[i,j]$ a $Q[i,j]$ pre x a y parciálne derivácie:

$$P[i,j] \approx (S[i,j+1] - S[i,j] + S[i+1,j+1] - S[i+1,j])/2 \tag{11}$$

$$Q[i,j] \approx (S[i,j] - S[i+1,j] + S[i,j+1] - S[i+1,j+1])/2 \tag{12}$$

Konečné rozdielnosti sú spriemerované podľa 2×2 štvorca aby x a y parciálne derivácie boli počítané pre ten istý bod obrazu. Magnitúdy a orientácia gradientu môže byť vypočítaná zo štandardných vzorcov pre converziu z karteziánskej súradnicovej sústavy do polárnej súradnicovej sústavy:

$$M[i,j] = \sqrt{P[i,j]^2 + Q[i,j]^2} \tag{13}$$

$$\theta[i,j] = \arctan(Q[i,j], P[i,j]) \tag{14}$$

kde funkcia arctan má dva argumenty a generuje uhol nad celou kružnicou možných smerov. Tieto funkcie musia byť počítané efektívne, bez používania floating-point aritmetiky. Je možné vypočítať

magnitúdy a orientáciu gradientu z parciálnych derivácií podľa tabuliek. Arcus tangens môže byť vypočítaný prevažne aritmetikou s pevnou rádovou čiarkou s niekoľkými floating-point výpočtami prevádzanými pomocou typu integer a aritmetiky s pevnou rádovou čiarkou.

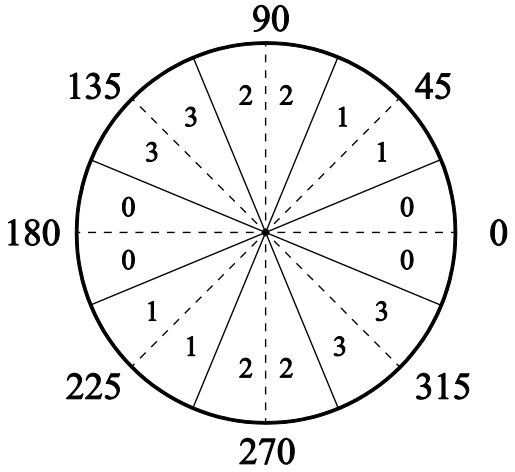
4.3.1 Nonmaxima Suppression (Potlačenie nemaximálnych hodnôt)

Pole magnitúd obrazu $M[i,j]$ bude mať veľké hodnoty kde je gradient obrazu veľký, ale nie dostatočný pre identifikáciu hrán, pretože problém nájdenia pozícií v poli obrazu, kde je rapídna zmena, bol transformovaný na problém hľadania pozícii v poli magnitúd $M[i,j]$, čo sú lokálne maximá. Pre identifikáciu hrán, hrany v poli magnitúd musia byť preriedené tak, že ostanú iba magnitúdy bodov pri najvačších lokálnych zmenách. Tento proces sa nazýva Nonmaxima Suppression (potlačenie nemaximálnych hodnôt), čo v tomto prípade vytvára preriedené hrany.

Potlačenie nemaximálnych hodnôt prieredi hrany gradient v $M[i,j]$ odstránením všetkých hodnôt pozdĺž línie gradientu, ktoré nie sú vrcholom hrebeňa hrany. Algoritmus začína redukovaním uhlu gradientu $\theta[i,j]$ na jeden zo štyroch sektorov zobrazených na obr. 4.3,

$$\zeta[i,j] = Sector(\theta[i,j]) \tag{15}$$

Algoritmus prevedie 3×3 *okolie* cez pole magnitúd $M[i,j]$. Pri každom bode, prostredný element $M[i,j]$ *okolia* je porovnaný s jeho dvomi susedmi pozdĺž línie gradientu daný sektorovou hodnotou $\zeta[i,j]$ v strede *okolia*. Ak hodnota poľa magnitúd $M[i,j]$ v strede nie je väčšia ako hodnota magnitúd oboch susedov pozdĺž línie gradientu, potom $M[i,j]$ je nastavené na nulu. Tento proces prieredi široké hrebene hrán gradientu magnitúd v $M[i,j]$ na hrebene len jeden pixel široké. Hodnoty výšky hrebeňa sú uložené v poli magnitúd potlačených nemaximálnych hodnôt.



Obr 4.3: Zobrazenie rozdelenia možných orientácií gradientu v sektoroch pre potlačenie nemaximálnych hodnôt. Obsahuje štyri sektory, číslované od 0 po 3, ktoré prislúchajú štyrom možným kombináciám elementov v 3×3 *okolí*. Línia teda musí prechádzať cez stred *okolia*. Rozdelenie kružnice možných orientácií línií gradientu je označené v stupňoch.

Nech

$$N[i,j] = nms(M[i,j], \zeta[i,j]) \tag{16}$$

značí proces potlačenia nemaximálnych hodnôt. Nenulové hodnoty v $N[i,j]$ korešpondujú k hodnote kontrastu v zmene intenzity obrazu. Napriek vyhladeniu prevedenému ako prvý krok detekcie hrán, pole magnitúd potlačených nemaximálnych hodnôt obrazu $N[i,j]$ bude obsahovať veľa falošných fragmentov hrán sposobených šumom a textúrou. Kontrast falošných fragmentov hrán je malý.

4.3.2 Thresholding (Prahovanie)

Typickou procedúrou používanou pre redukciu falošných fragmentov hrán v poli magnitúd potlačených nemaximálnych hodnôt gradientu je použitie prahu pre $N[i,j]$. Všetky hodnoty pod prahom sú zmenené na nulu. Výsledok použitia prahovania pre pole magnitúd potlačených nemaximálnych hodnôt je pole hrán detekovaných v obraze $I[i,j]$. Stále sa budú vyskytovať aj nejaké falošné hrany, pretože prah τ bol príliš malý, alebo niektoré kontúry môžu chýbať kvôli zjemneniu hranového kontrastu tieňmi alebo kvôli príliš vysokému prahu τ. Výber vyhovujúceho prahu je obtiažne a vyžaduje množstvo pokusov.

Algoritmus dvojitého prahovania aplikuje na pole magnitúd potlačených nemaximálnych hodnôt $N[i,j]$ dva prahy τ_1 a τ_2, $\tau_2 \approx 2\tau_1$, pre vytvorenie dvoch obrazov s detekovanými hranami $T_1[i,j]$ a $T_2[i,j]$. Keďže obraz T_2 bol vytvorený vyšším prahom, bude obsahovať menej falošných hrán; ale T_2 môže mať diery v kontúrach. Algoritmus dvojitého prahovania spojí hrany v T_2 do kontúr. Keď dosiahne koniec kontúry, algoritmus hľadá hrany na pozíciách 8 susedov v T_1. Algoritmus pokračuje v premosťovaní diery až pokiaľ dosiahne hranu v T_2. Tento postup rieši niektoré problémy s výberom prahu.

4.3.3 Algoritmus Canny hranového detektora

1. *Vyhladenie obrazu Gaussovým filtrom.*
2. *Výpočet magnitúd a orientácie gradientu.*
3. *Aplikácia potlačenia nemaximálnych hodnôt na pole magnitúd gradientu.*

4. *Použitie dvojitého prahovania pre detekciu a prepojenie hrán.*

4.4 Sobel operátor

Sobel operátor je jeden z najpoužívanejších hranových detektorov. Pre výpočty gradientu je použité 3×3 *okolie*. Uvažujme rozloženie pixlov okolo pixla $[i,j]$ podľa obrázka 4.4. Sobel operátor je magnitúda gradientu vypočítaná podľa

$$M = \sqrt{s_x^2 + s_y^2} \tag{17}$$

kde sú parciálne derivácie počítané podľa

$$s_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \tag{18}$$

$$s_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4) \tag{19}$$

s konštantou $c = 2$.

Ako pri iných gradientových operátoroch, s_x a s_y môžu byť implementované použitím konvolučných masiek:

$$s_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad s_y = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} \tag{20}$$

Tento oberátor kladie dôraz na pixle, ktoré sú bližšie v stredu masky [4].

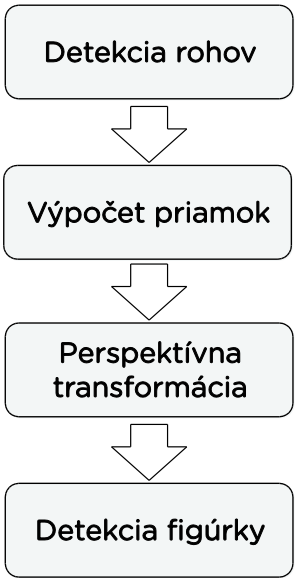
a_0	a_1	a_2
a_7	[i,j]	a_3
a_6	a_5	a_4

Obr 4.4

5 Návrh systému

Prvotný návrh zahŕňal použitie funkcie pre detekciu šachovnice z knižnice OpenCV – `findChessboardCorners` . Táto funkcia sa však používa prevažne pre kalibráciu kamery a neposkytuje požadované výsledky pri detekcii šachovnice s figúrkami. Z toho dôvodu bolo potrebné navrhnúť aj spôsob detekcie šachovnice.

Navrhnutý systém teda pozostáva z niekoľkých modulov, z ktorých je každý závislý na predošlom module (obrázok 5.1). Modul detekcie rohov rozpoznáva rohy šachovnice a vytvára pole detekovaných rohov. Toto pole je následne prebrané modulom detekcie a výpočtu šachovnicových priamok. V module perspektívnej transformácie šachovnice sa vypočítajú súradnice jednotlivých políčok šachovnice a prevedie sa perspektívna transformácia. Modul detekcie figúrky následne určí, či dané políčko obsahuje bielu alebo čiernu figúrku, alebo je prázdne.



Obr 5.1

5.1 Detekcia rohov šachovnice

Modul detekcie rohov šachovnice v prvom kroku aplikuje konverziu farebného priestoru (viď 4.1) a prahovanie (viď 4.2) s prahom daným paramterom $THRESH\_OTSU$ funkcie OpenCV $Imgproc.treshold$ na spracovávaný obraz. Takýto obraz (obr. 4.2) je pripravený pre algoritmus hľadania potencionálnych rohov šachovnice.

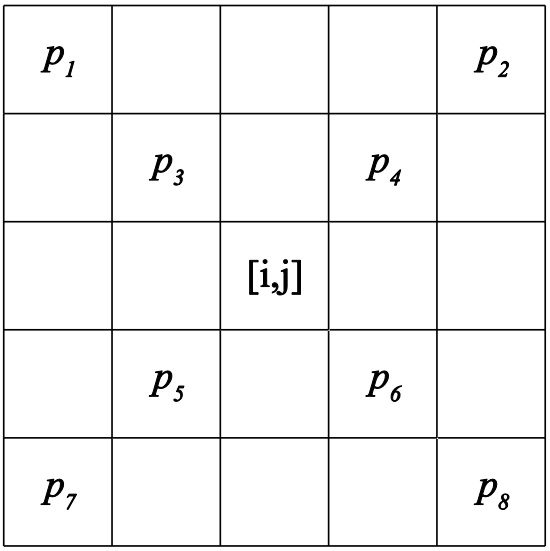
Princíp hľadania potencionálnych rohov šachovnice je inšpirovaný postupom popísaným v kapitole Sampling Strategy článku ChESS – Quick and robust detection of chess-board feature [5].

Uvažujme rozloženie pixlov podľa obr. 5.3.

Pre každý pixel obrazu ak platí

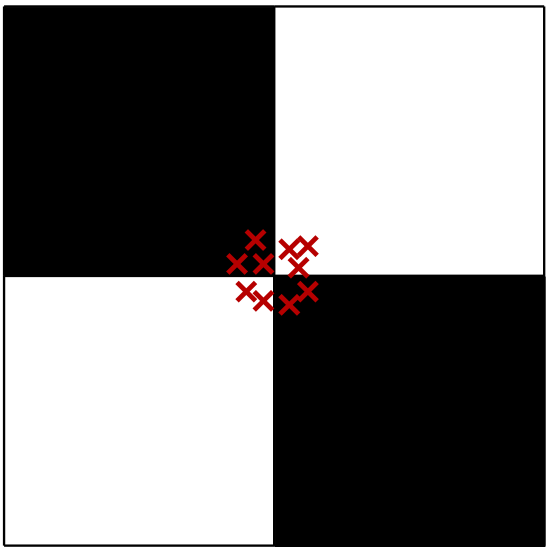
$$(p_1 = p_3 = p_6 = p_8)\Lambda(p_2 = p_4 = p_5 = p_7)\Lambda(p_1 \neq p_2) \quad (21)$$

potom je pixel zaradený do poľa potencionálnych rohov P.



Obr 5.3

Ak bol uložený pixel pixlom skutočnéno rohu šachovnice, okolo daného rohu bude detekované vačšie množstvo podobných pixlov, teda akýsi zhluk bodov (obr.5.4). Táto vlastnosť neskôr pomôže odfiltrovať falošné rohy, ktoré môžu byť detekované na podobnom mieste textúry obrazu, kde sa vyskytne podobné striedanie farieb ako u šachovnicového rohu.



Obr 5.4 – ilustrácia zhluku detekovaných bodov

Výpočet skutočných rohov berie pole potencionálnych rohov P, z ktorého pre všetky body aplikuje nasledujúci postup. Vyberie konkrétny bod z poľa potencionálnych rohov a uloží do dočasného poľa tmp. Tento bod označíme ako $Core$. Pre všetky ostatné body P vypočíta vzdialenosť potencionálneho bodu od $Core$. Ak je vzdialenosť menšia ako daný polomer r, bod je uložený do tmp a odstránený z P. Po dokončení prechádzania P pre daný $Core$ bod, sa porovná počet bodov v tmps určeným prahom τ. Ak je počet menší, tento prah odfiltruje falošné potencionálne rohy, inak vypočíta strednú pozíciu bodov tmp. Výslednú pozíciu uloží ako korektne detekovaný roh, vyprázni tmp a pokračuje nasledujúcim $Core$ bodom.

Pseudokód:

```
1.  pre všetky body v P
2.  {  core = P[i, j];
3.     tmp ← core
4.     pre všetky ostatné body v P
5.     {
6.           if (vzdialenosť core od P[k, l]) < r
7.           {     tmp ← P[k, l]
8.                 odstráň P[k, l] z P }
9.     }
10.    if (počet bodov v tmp) > τ
11.    {
12.          // vypočítať strednú pozíciu z bodov v tmp
13.          // uložiť do výsledného poľa
14.    }
```

5.2 Výpočet priamok šachovnice

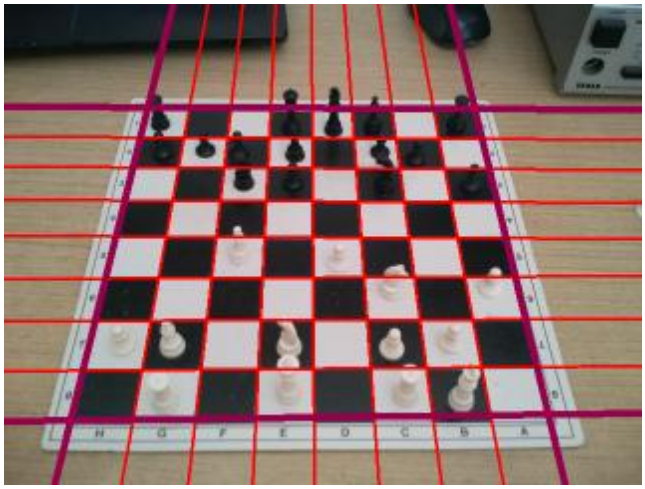
Program pokračuje výpočtom šachovnicových priamok ak je počet detekovaných rohov nenulový, čo však nezaručuje úspešnú detekciu mriežky šachovnice. Pre výpočet priamky sa určí najmenšia vzdialenosť d dvoch rohov z poľa detekovaných rohov a vyberie konkrétny roh šachovnice. K tomuto rohu sa z ostatných rohov vyberú najbližší susedia, ktorých vzdialenosť od pôvodného rohu je menšia ako $2d$. Ak je počet susedov nenulový, vyberie sa jeden sused, a pomocou smerového vektora sa hľadá ďalší sused v danej línii. Neúspešné hľadanie dvoch susedov v línii vedie k výberu nasledujúceho rohu a opakuje celź postup. Ak je hľadanie úspešné, program našiel tri rohy jednej

priamky šachovnice, a pokračuje procedúrou hľadania ostatných rohov danej línie z poľa detekovaných rohov. Hľadanie je úspešné ak nájde všetkých sedem rohov danej priamky, inak opakuje celý postup pre ďalší roh, alebo končí program neúspechom po vyčerpaní všetkých detekovaných rohov.

Po detekovaní siedmych rohov v jednej línii je detekovaná prvá kompletná priamka p šachovnice. Algoritmus pokračuje nachádzaním kvázi kolmíc na danú priamku p vedených každým bodom priamky p. Postup je podobný ako pri detekovaní pôvodnej línie. Pre každý bod sa nájdu najbližší susedia, ktorých smerový vektor zviera so smerovým vektorom priamky p určitý uhol $\propto$. Pri tomto uhle je potrebné rozhodnúť, či sú vektory rovnobežné, alebo kvázi kolmé. Detekcia susedov s kvázi kolmými smerovými vektormi pre každý bod priamky p určuje smery ďalších siedmych priamok šachovnice. Následne sa ostatné detekované rohy priradia kvázi kolmým priamkám podľa smerových vektorov a spätne sa dopočítajú jednotlivé rovnobežky k priamke p podľa bodov kompletných kvázi kolmých priamok (obr. 5.5).

Obr 5.5 – ukážka detekovaných priamok; bodové značenie pre priamky poľa č. 1, kružnicové značenie pre priamky poľa č. 2

Výsledok detekcie priamok je dvojica polí obsahujúcich sedem kvázi rovnobežných priamok. Chýbajú však okrajové priamky, ktoré neboli detekované pomocou šachovnicových rohov. Pre tento problém je treba pre každú detekovanú priamku vypočítať priesečník s okrajmi obrazu. Podľa súradníc priesečníkov sa určia aktuálne okrajové priamky a priamky, ktoré sú okrajové ako druhé v poradí. Rozdielom súradníc takýchto dvojíc priamok sa jednoducho dopočítajú okrajové priamky šachovnice, pričom sa doplnia do korešpondujúceho poľa priamok podľa ich smeru. Výsledné polia potom obsahujú deväť kvázi rovnobežných priamok (obr. 5.6).



Obr 5.6 – detekované priamky šachovnice doplnené o okrajové priamky

5.3 Perspektívna transformácia

Pre analýzu jednotlivých políčok šachovnice je potrebné vypočítať súradnice ich rohov, ktoré odpovedajú priesečníkom jednotlivých priamok v rámci obrazu.

Priesečník dvoch priamok L_1 a L_2 v rovine s L_1 prechádzajúcou bodmi (x_1, y_1) a (x_2, y_2) a L_2 prechádzajúcou bodmi (x_3, y_3) a (x_4, y_4) je daný

$$x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}} \tag{22}$$

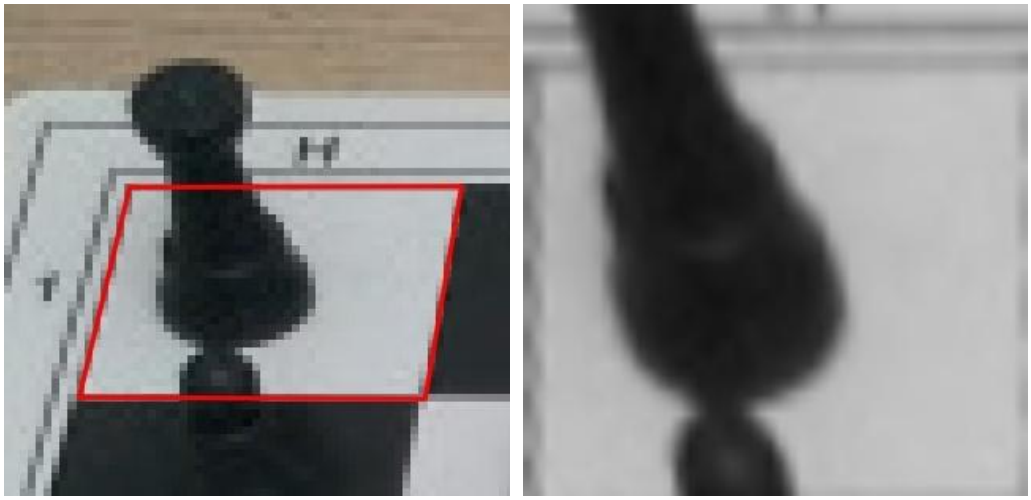
$$y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}} \tag{23}$$

kde $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$ znamená determinant [6].

Vypočítané súradnice jednotlivých políčok umožnia vykrojiť konkrétne políčko, previesť perspektívnu transformáciu a detekovať figúrku.

Keďže je obraz snímaný z perspektívy, niektoré časti snímaného objektu, ktoré sú bližšie k snímaciemu zariadeniu, sú vykreslené väčšie ako časti, ktoré sú vzdialenejšie od kamery. Obraz je

tak perspektívne deformovaný. Z toho dôvodu je potrebné previesť perspektívnu transformáciu, ktorá prevádza 3D svet do 2D podoby. V našom prípade je transformácia aplikovaná na jednotlivé políčka, čoho výsledkom je prevod nepravidelného štvoruholníka na obdĺžnik ako na obr. 5.7. Perspektívna transformácia je prevedená pomocou funkcie knižnice OpenCV - `warpPerspective`.



Obr 5.7 – perspektívna transformácia

5.4 Detekcia figúrky

Pre detekciu figúrky bolo použitých niekoľko rôznych algoritmov. Základom sú hranové detektory Canny viď. 4.3 a Sobel operátor viď. 4.4.

5.4.1 Sobel operátor

Uvažujme políčko šachovnice po vykrojení a prevedení perspektívnej transformácie ako na obr. 5.7. Algortimus aplikuje konverziu farebného priestoru a prevod obrazu na čiernobiely pre pripravenie matice obrazu políčka *S* na detekciu hrán. Použitie Sobel operátora využitím funkcie knižnice OpenCV – `Sobel` je predvedené na obr. 5.8.

Obr 5.8 – použitie Sobel operátora

Kvôli nie stopercentnej presnosti pri určení pozícii rohov sa môžu v obraze vyskytnúť rušivé elementy na krajoch obrazu políčka. Algoritmus preto môže začínať prechádzanie obrazu po riadkoch zhora nadol s určitým vertikálnym aj horizontálnym posunom, pričom tiež môže končiť prechod v určitej vzdialenosti od spodného a pravého okraja obrazu.

Algoritmus detekcie figúrky očakáva detekovanie okraja figúrky zľava aj zprava. Počiatočný a koncový bod horizontálnych úsekov ohraničených kontúrami figúrky sú uložené do poľa kontúr $C..$ V prípade nájdenia takýchto úsekov ráta počet pixlov nad a pod určitým prahom, podľa čoho určí farbu danej figúrky. V tomto prípade je detekcia hrán figúrky dosiahnutá len z pravej strany, program rozhodne o analyzovaní prázdneho políčka a detekcia je neúspešná. Detekovanie obrysu figúrky závisí od prahu τ konverzie obrazu na čiernobiely, kvality obrazu, jasu, tieňov a iných faktorov.

Pseudokód:

1. preveď prevod obrazu S do odtieňov šedej
2. preveď prevod S na obraz čiernobiely s prahom τ
3. aplikuj Sobel operátor na S
4. **pre všetky riadky** S
5. **if**(existujú minimálne dva biele body v riadku)
6. $C \leftarrow$ dvojica najľavejšieho a najpravejšieho bieleho bodu v riadku
7. **if**(veľkosť poľa $C >$ prah počtu dvojíc kontúr)
8. **pre všetky dvojice z** C
9. spočítaj pixle s hodnotou nad a pod prahom v riadku medzi dvojicou
10. **if**(čierne $<$ biele) biela figúrka
11. **else** čierna figúrka
12. **else** prázdne políčko

5.4.2 Canny hranový detektor

Podobne ako v predošlom postupe, uvažujeme políčko S podľa obr. 5.7. Použitý je detektor hrán Canny s využitím funkcie knižnice OpenCV – `Canny`. Po aplikovaní detektoru hrán je detekovaný vyhovujúci obrys detekovanej figúrky (obr. 5.9).

Obr 5.9 – použitie detektoru Canny

Princíp detektora je podobný ako v predošlom detektore viď. 5.4.1. Líši sa vo vynechaní konverzie obrazu na čiernobiely a v použitom prahu τ, ktorý je v tomto detektore parametrom funkcie Canny.
V prípade obr. 5.9 program rozhodne o analyzovaní políčka obsahujúceho čiernu figúrku.

Pseudokód:

1. preveď prevod obrazu S do odtieňov šedej
2. aplikuj Canny funkciu s prahom τ
3. **pre všetky riadky** S
4. **if**(existujú minimálne dva biele body v riadku)
5. $C \leftarrow$ dvojica najľavejšieho a najpravejšieho bieleho bodu v riadku
6. **if**(veľkosť poľa C $>$ prah počtu dvojíc kontúr)
7. **pre všetky dvojice z** C
8. spočítaj pixle s hodnotou nad a pod prahom v riadku medzi dvojicou
9. **if**(čierne $<$ biele) biela figúrka
10. **else** čierna figúrka
11. **else** prázdne políčko

5.4.3 Canny hranový detektor a histogram

Pri predošlých detektoroch sa spolieha na kvalitný obraz s výraznými hranami, ktoré by jednotlivé algoritmy pri vhodne zvolených prahoch rozlišovania hrán dokázali rozlíšiť. Kedže väčšina prípadov nie je na toľko dokonalá, bolo potrebné navrhnúť aj systém, ktorý by riešil aj menej kvalitné až splývajúce obrazy. To bolo z väčšej časti dosiahnuté kombináciou Canny detektoru s histogramom odtieňov šedej.

Upravený algoritmus vytvára histogram farieb H, ktorý je tvorený počtom pixlov jednotlivých odtieňov šedej. Odtiene sa spočítajú na základe určeného prahu τ. V prípade dostatočného počtu odtieňov jednej farby pod prahom δ' a druhej farby nad prahom δ'', program rozhodne o analýze políčka obsahujúceho čiernu, resp. bielu figúrku. Ak je počet odtieňov príliš malý a ich suma vysoká (b), program rozhodne o analýze prázdneho políčka.

V prípade, že je počet odtieňov vyvážený, program aplikuje Canny detektor.

Pseudokód:

1. preveď prevod obrazu S do odtieňov šedej
2. **pre všetky body** S
3. vytvor histogram odtieňov farieb H
4. **pre všetky záznamy v** H
5. **if**(hodnota $> \tau$)
6. čierne++ , suma_čiernych += hodnota
7. **else** biele++ , suma_bielych += hodnota
8. **if**((čierne $< \delta'$ && suma_čiernych $> b$) alebo (biele $< \delta'$ && suma_bielych $> b$))
9. prázdne políčko
10. **else if**((čierne $> \delta'$ && biele $< \delta''$) čierna figúrka
11. **else if**((biele $> \delta'$ && čierne $< \delta''$) biela figúrka
12. **else**
13. aplikuj Canny funkciu
14. **pre všetky riadky** S
15. **if**(existujú minimálne dva biele body v riadku)
16. $C \leftarrow$ dvojica najľavejšieho a najpravejšieho bieleho bodu
17. **if**(veľkosť poľa C $>$ prah počtu dvojíc kontúr)
18. **pre všetky dvojice z** C
19. spočítaj pixle s hodnotou nad a pod prahom medzi dvojicou
20. **if**(čierne $<$ biele) biela figúrka
21. **else** čierna figúrka
22. **else** prázdne políčko

6 Implementácia

Úlohou tejto bakalárskej práce bolo vytvoriť funkčnú aplikáciu na mobilnom zariadení. Zvolenou platformou pre tento program bol Android. Počas vytvárania aplikácie boli informácie o vývoji a postupy čerpané z developerských stránok Androidu http://developer.android.com/.

6.1 Výpočtová časť

Výpočtová časť programu bola realizovaná podľa návrhu systému popísaného v sekcii 5. Vytvorená bola riadiaca activity s jedným fragmentom. Fragment obsahuje tlačidlá pre výber fotografie z galérie, detekciu figúrok a opravu pozície na výslednej šachovnici.

Tlačidlo detekcie šachovnicových figúrok vytvára objekt triedy `ChessboardRecognition`, ktorá sa stará o kompletný proces detekcie po obdržaní bitmapy vybraného obrazu. Objekt tejto triedy obsahuje hlavné metódy pre detekciu potencionálnych rohov, detekciu a výpočet korektných rohov, detekciu a výpočet priamok šachovnice, výpočet okrajových priamok šachovnice, výpočet súradníc rohov jednotlivých políčok šachovnice, a metódu pre detekciu figúrky na konkrétnom políčku.

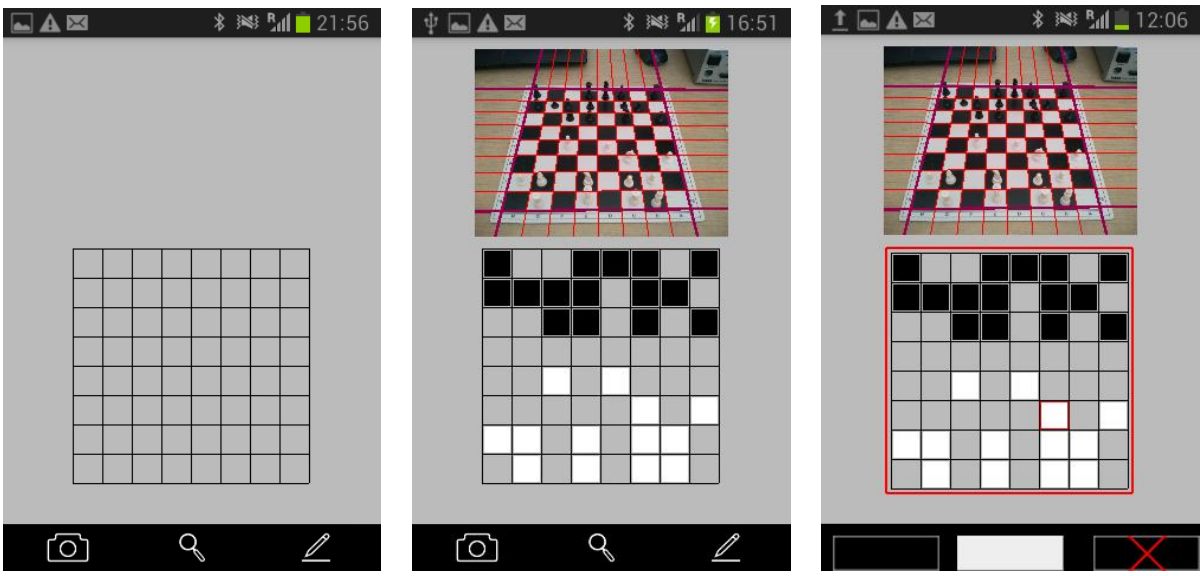
Kedže algoritmy pre detekciu figúrok popísané v sekcii 5.4 nie sú stopercentne úspešné, funkcia opravy chyby je v aplikáciách podobného typu významná a užitočná. Z toho dôvodu bola implementovaná táto funkcia v podobe tlačidla opravy pozície na výslednej šachovnici. Toto tlačidlo aktivuje možnosť kliknutia na detekovanú šachovnicu, pričom po kliknutí zobrazuje pop-up okno s ponukou výberu čiernej figúrky, bielej figúrky a prázdneho políčka.

Pri vývoji aplikácie boli využité funkcie knižnice OpenCV najnovšej verzie dostupné na http://sourceforge.net/projects/opencvlibrary/files/opencv-android/. Inštalácia a import balíka boli prevedené poďla návodov z oficiálnych stránok http://opencv.org/platforms/android.html.

6.2 Prostredie aplikácie

Užívateľské prostredie programu je navrhnuté pre čo najjednoduchšiu a intuitívnu obsluhu. Všetky ovládacie prvky sú po celý čas behu prgramu zobrazené v podobe lišty v spodnej časti obrazovky, kde sú zreteľne viditeľné a rýchlo dostupné pre pohodlnú prácu s aplikáciou. Prostredie programu tvorí jediný fragment, ktorý sa stará o zobrazenie tlačidiel, spracovávaného obrazu, detekovaných priamok

šachovnice a samotného výsledku detekcie. Ten je realizovaný vykreslením mriežky 8×8 políčok, pričom detekcia čiernej a bielej figúrky je prevedená zobrazením vyplneného štvorca odpovedajúcej farby na danej pozícii v prípade korektnej detekcie konkrétnej figúrky. Prostredie aplikácie je zobrazené na obr. 6.1.



Obr 6.1 – užívateľské prostredie aplikácie:

a) vľavo pred výberom obrazu

b) v strede po detekcii

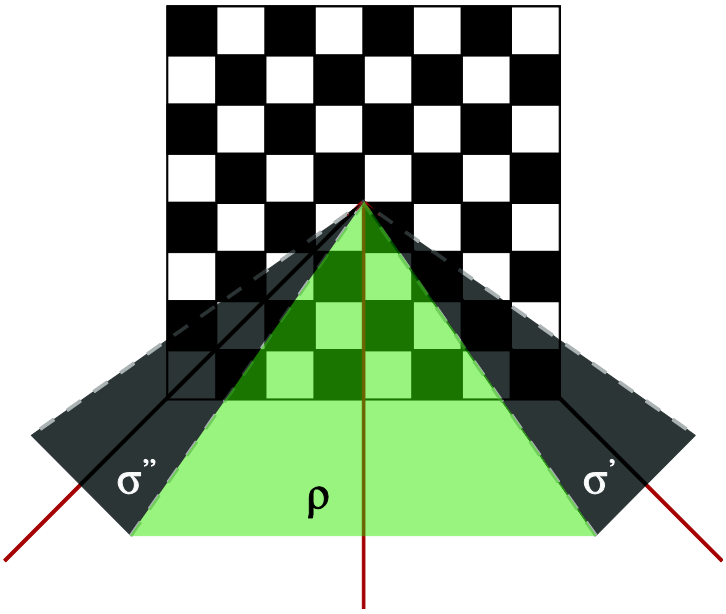
c) vpravo ponuka opravy pozície

7 Vyhodnotenie

Program obsahuje dva typy implementovaných detektorov. Detektor šachovnice a detektor figúrok.

7.1 Detektor šachovnice

Detektor šachovnice vyžaduje úspešné detekovanie aspoň dvoch kompletných sedem-bodových línii, z čoho potom dokáže vypočítať priamky šachovnice. Úspešnosť detekcie sa pohybuje okolo 70%, pričom je ovplyvnená rôznymi faktormi. Kvalita fotografie je jedným z nich, no hlavným faktorom je uhol pohľadu na analyzovanú fotografiu. V prípade prekrývania rohov šachovnice figúrkami sa úspešnosť detekcie šachovnice znižuje, no výrazný pokles korektnej detekcie spôsobuje $45°$ uhol pohľadu od kolmice na stred šachovnice v horizontálnom smere v rozmedzí približne $\pm 10°$ (obr. 7.1). To je spôsobené detekciou rohov, ktorá ráta s rohmi približného tvaru $+$.



Obr 7.1 – Vplyv pohľadu na šachovnicu na detekciu rohov

a) ρ – úspešná detekcia väčšiny prípadov

b) σ´ a σ´´ – neúspešná detekcia rohov

7.2 Detektory figúrok

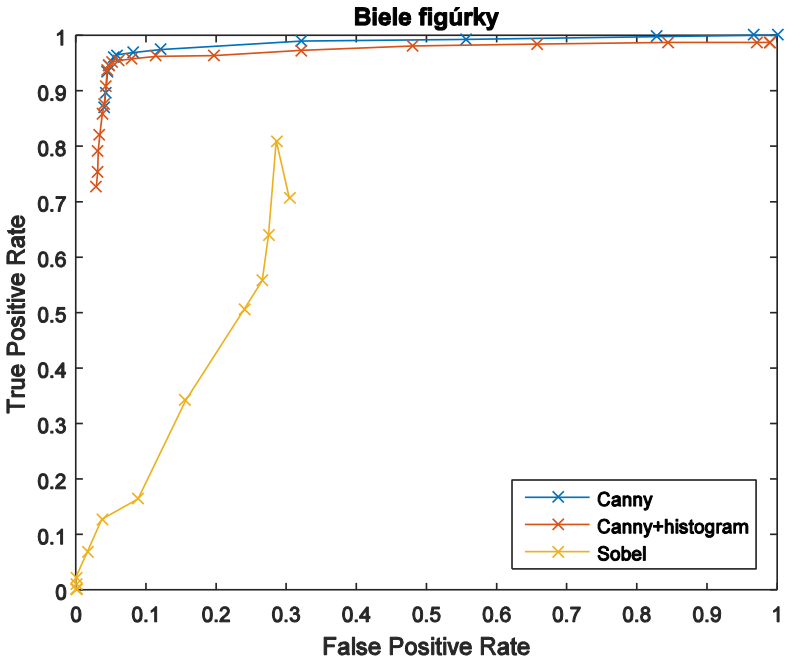
V aplikácii boli implementované tri druhy detektorov figúrky. Na vyhodnotenie úspešnosti boli použité ROC (receiver operating characteristic) krivky (obr. 7.2 a obr. 7.3), kde vertikálna os označuje $true\ positive\ rate$ a horizontálna os označuje $false\ positive\ rate$. Jednotlivé hodnoty boli vypočítané podľa vzorcov

$$TPR \approx \frac{positives\ \ correctly\ \ classified}{total\ \ positives} \tag{24}$$

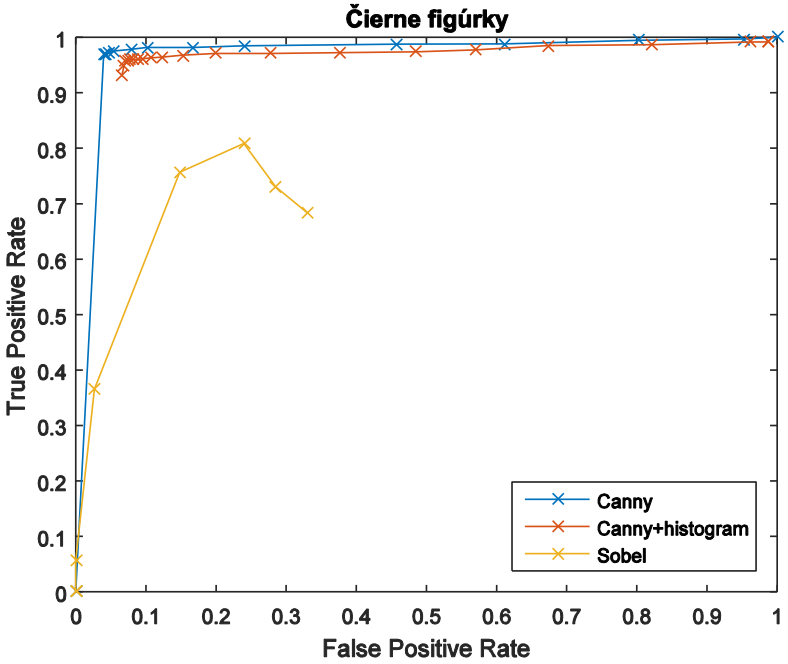
$$FPR \approx \frac{negatives\ \ incorrectly\ \ classified}{total\ \ negatives} \tag{25}$$

kde $positives\ correctly\ classified$ označuje počet správne detekovaných figúrok, $total\ positives$ označuje počet figúrok na konkrétnej šachovnici, $negatives\ incorrectly\ classified$ označuje počet nesprávne detekovaných figúrok a $total\ negatives$ označuje počet prázdnych políčok na konkrétnej šachovnici [7].

Pri testovaní a vyhodnocovaní boli u každého detektoru upravované určité prahy vplývajúce na výsledok detekcie. Konkrétne vstupný parameter OpenCV funkcie `Canny` u Canny detektora, prah prevodu obrazu na čiernobiely pre Sobel detektor a hranica rozlišovania či políčko obsahuje figúrku alebo je prázdne pre Canny detektor kombinovaný s histogramom.



Obr 7.2 – Graf úspešnosti detekcie bielych figúrok jednotlivých detektorov



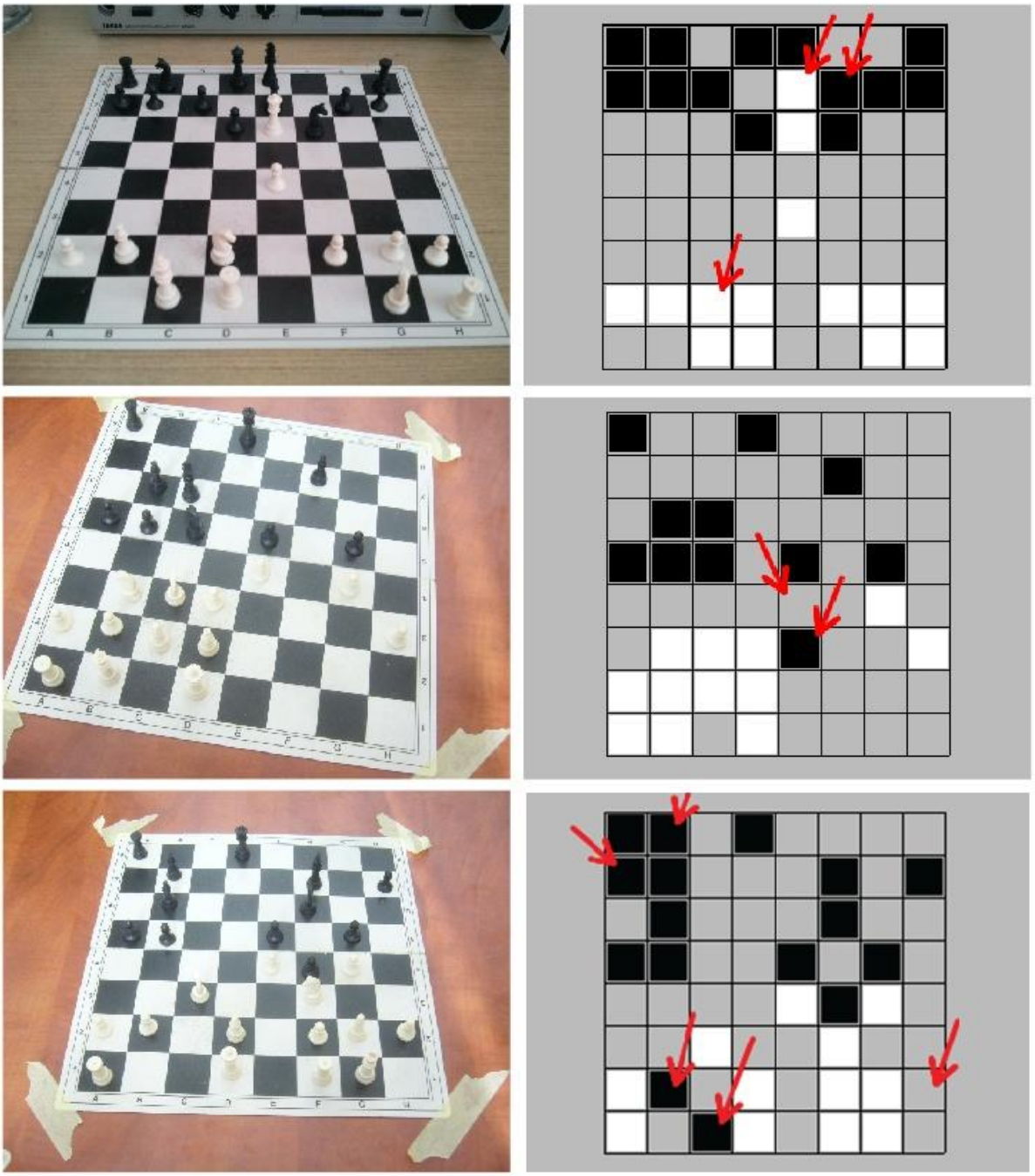
Obr 7.3 – Graf úspešnosti detekcie čiernych figúrok jednotlivých detektorov

V grafoch je vidieť náhly pokles úspešnosti detekcie u Sobel detektora. To je spôsobené tým, že po prekročení určitej hodnoty prahu prevodu obrazu na čiernobiely sa farby niektorých figúrok prekonvertujú na farby opačné.

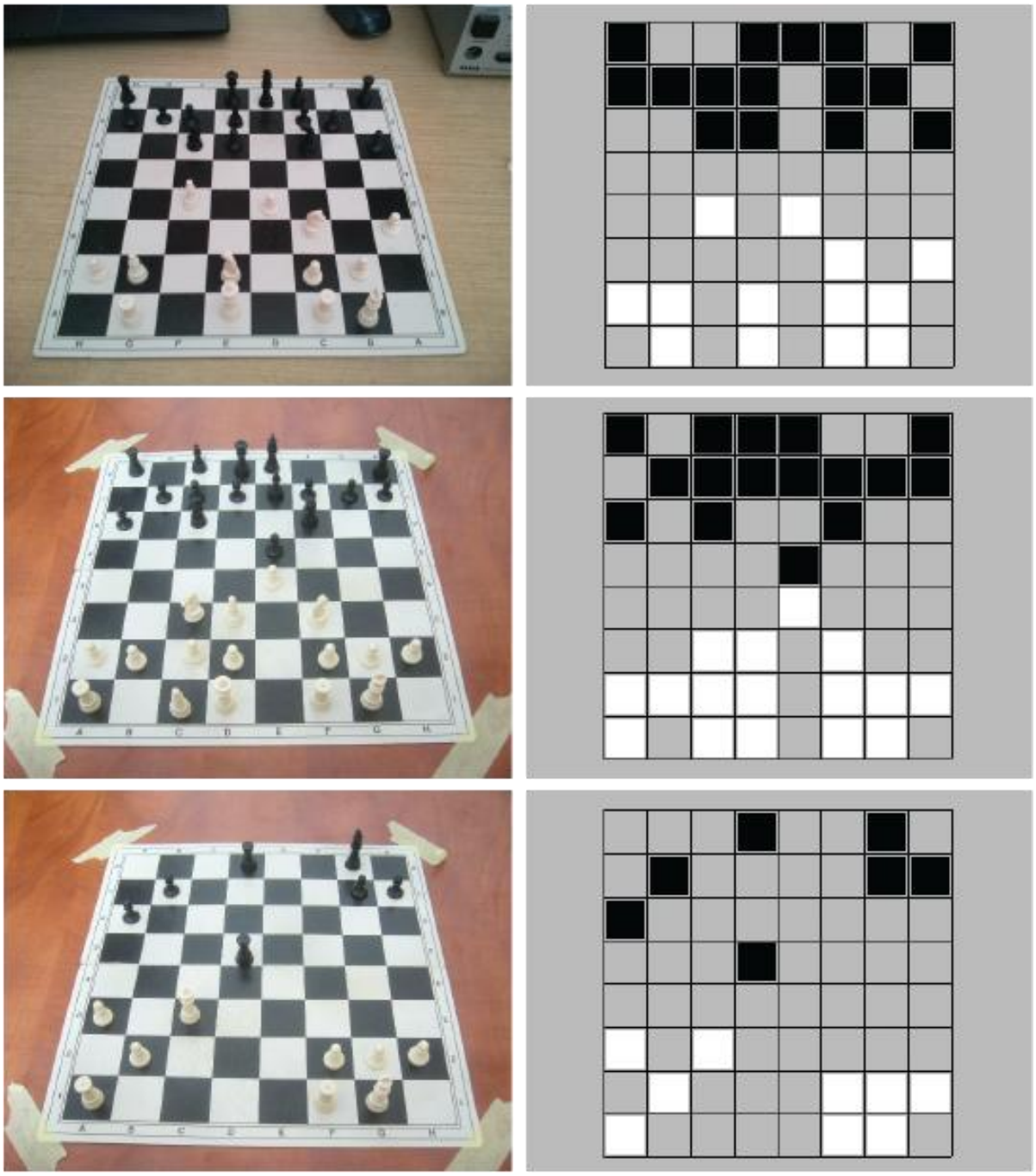
Detektor založený na Sobel operátore rapidne zlyháva pri detekovaní prípadu rovnakej farby políčka aj ním obsahujúcej figúrky (obr. 7.4.). To je spôsobené nedostatočnou kvalitou obrazu, alebo málo výraznými až splývajúcimi hranami. Naopak z grafov vidieť, že Canny hranový detektor a kombinovaný Canny detektor s histogramom sú výrazne úspešnejší ako predchádzajúci detektor.

Obr 7.4 – Ukážka zlyhania Sobel detektoru
Zľava: a) obraz bieleho políčka obsahujúceho bielu figúrku;
b) obraz po aplikácii Sobel operátora

Detekcia figúr najčastejšie zlyháva na políčkach, do ktorých zasahujú vysoké figúrky z políčka pod ním. Záleží na pozorovacom uhle, do akej miery budú figúrky prekrývať ostatné políčka. Ďalšie zlyhania spôsobujú tiene, figúrky postavené príliš blízko okraja políčka alebo nekvalitný fragment fotografie. Jednotlivé prípady detekcií a zlyhaní sú zobrazené na obr. 7.5. Obr. 7.6 zobrazuje bezchybne detekované šachovnice.



Obr 7.5 – Detekované prípady a zvýraznené chyby detekcie (použitý Canny detektor s histogramom)



Obr 7.6 – Bezchybne detekované prípady

7.3 Možné úpravy a rozšírenia

Aplikácia pracuje s obrázkami, ktoré sú prevádzané do rozsiahlych matíc, čo je dôvodom náročných výpočtov. Doba spracovania jednej fotografie sa pohybuje okolo 12 sekúnd, čo však záleží na veľkosti obrazu (testované rozlíšenie bolo 640×480 pixlov) a výpočtovom výkone zariadenia, na ktorom aplikácia pobeží.

7.3.1 Detektor rohov

Pre zvýšenie úspešnosti detekcie šachovnice by bolo vhodné upraviť algoritmus detekcie rohov tak, aby rozpoznával aj rohy blízke tvaru $\times$. Vyžadovalo by to však ďalšie dôkladné ladenie a testovanie aj kvôli vplyvu na celkový výpočtový čas procesu detekcie, pretože z veľkej časti (približne polovica času) zaberá práve detekcia rohov.

7.3.2 Real-time detekcia

Zaujímavým rozšírením aplikácie by mohla byť real-time detekcia. V programe by bola funkcia pre zapnutie pohľadu cez kameru, pričom aplikácia by počas snímania detekovala jednotlivé figúrky, prípadne len šachovnicu s možnosťou voľby zachytenia konkrétneho pohľadu a dokončenia operácie detekcie.

7.3.3 Rozpoznanie typu figúrok

Aplikácia v aktuálnom stave dokáže rozpoznať pozície a farbu jednotlivých figúrok na šachovnici, čo má však isté obmedzenia v praktickom použití programu. Vhodným rozšírením by bolo pridanie manuálnej voľby typu konkrétnej figúrky, alebo úprava aplikácie tak, aby automaticky rozlíšila typ figúrky, pričom by detekciu bolo možné založit na multi-view detektore.

8 Záver

Výsledkom tejto bakalárskej práce je funkčná aplikácia detekcie figúrok na fotografii šachovnice. Aplikácia bola vytvorená pre platformu Android a využíva funkcie najnovšej verzie knižnice OpenCV. Detekcia prebieha v dvoch hlavných krokoch, detekcii šachovnice a detekcii figúr. Pre detekciu figúr boli použité tri algoritmy a výsledkom procesu je zobrazenie pozícií a farieb jednotlivých šachových figúrok. Úspešnosť detekcie bola vyjadrená ROC (receiver operating characteristic) krivkami a závisí od viacerých faktorov, napr. od kvality fotografie, uhlu pohľadu na šachovnicu alebo zvoleného prahu pri rozlišovaní figúrok. Úspešnosť detekcie figúrok a ich farieb jednotlivých detektorov sa pohybuje okolo 89% pri detektore Canny s histogramom, 83% pri detektore Canny a 24% pri detektore Sobel. Priemerná úspešnosť detekcie šachovnice je približne 70%.

Zadanie práce zahŕňa štúdium algoritmov pre rozpoznanie šachovnice a figúr na šachovnici, návrh systému detekcie figúrok na šachovnici a jeho implementácia na zvolenú mobilnú platformu a vyhodnotenie úspešnosti rozpoznania pozícií. Jednotlivé požiadavky zadania boli splnené tak, že boli naštudované jednotlivé techniky detekcií, bola navrhnutá a implementovaná aplikácia na platformu Android a výsledok detekcie bol popísaný v kapitole 7. K práci bol vytorený aj prezentačný plagát a video.

Aplikácia poskytuje možnosti prípadného rozšírenia v podobe real-time detekcie, rozpoznávania typu figúrok alebo iných úprav.

Literatura

1. **Zhanyang, Z, a iní.** An Inproved Corner Detection Algorithm Based On Harris. *Advanced Engineering Forum.* 2012, Zv. 6, 7, s. 717 - 721.

2. **Weixing, Z., a další.** *A Fast and Accurate Algorithm for Chessboard Corner Detection.* Tianjin : 2nd International Congress on Image and Signal Processing, 2009. stránky 1 - 5. ISBN 978-1-4244-4129-7.

3. **Johnson, Stephen.** *Stephen Johnson on Digital Photography.* Taliansko : O'Reilly, 2006. ISBN 0-596-52370-X.

4. **Jain, R., Kasturi, R. a Schunck, B. G.** *Machine Vision.* New York, USA : McGraw-Hill, 1995. ISBN: 0-07-032018-7.

5. **Bennett, S. a Lasenbly, J.** ChESS – Quick and robust detection of chess-board features. *Computer Vision and Image Understanding.* 2014, Sv. 118, stránky 197-210.

6. **Weisstein, Eric W.** Line-Line Intersection. *MathWorld.* [Online] A Wolfram Web Resource. [Citace: 28. 4 2015.] http://mathworld.wolfram.com/Line-LineIntersection.html.

7. **Fawcett, Tom.** An Introduction to ROC Analysis. *Pattern Recognition Letters.* 2006, Sv. 27, 8, stránky 861-874.